COOK’S THEOREM

1. NP = P is the satisfiability problem is a P problem
2. SAT is NP problem
3. It’s a first NP-complete problem
4. Every NP problem reduces to SAT

-> A decision problem is in NP if it can be solved by a non-deterministic
algorithm in polynomial time.
-> An instance of the Boolean satisfiability problem is a Boolean expression that combines Boolean variables using Boolean operators.
-> An expression is satisfiable if there is some assignment of truth values to the variables that makes the entire expression true.

PROOF

1. SAT is in NP
2. Convert the execution of polynomial time Nondeterministic Turning Machine to a bunch of well formed formulae such that it satify, if the machine accepts the input
3. Show the sum of lengths of formulae is polynomial in the size of prolem
4. NP-Hard - Can polynomially reduce any NP problem to L
5. NP-Complete LENP
6. LENP -> NDTM for that runs in polynomial time
7. An NDTM is the only model we’ve for NP problem
8. SAT $\in$ NP
9. Therefore, if we can polynomially reduce an arbitrary polynomial
NDTM to SAT, it means we’ve proven SAT is NP-Complete,
proving Cook’s Theorem

Savitch’s Theorem

In Automata, Savitch's theorem, proved by Walter Savitch in 1970, gives a relationship between deterministic and non-deterministic space complexity. It states that for any function fε Ω(log(n)),

$$NSPACE(f(n)) \subseteq DSPACE(f(n)2).$$

In other words, if a nondeterministic turing machine can solve a problem using f(n) space,then an ordinary deterministic turing machine can also solve the same problem in the square of that space bound.Although it seems that nondeterminism may produce exponential gains in time, this theorem shows that it has a markedly more limited effect on space requirements.

Proof:

Suppose T is a non-deterministic Turing machine which requires at most f(n) space on any input for some f≥n.
We will describe a deterministic Turing machine which requires O(f(n)2) space and has the same behavior as T. We first define the procedure CANYIELD, which given two configurations c1 and c2 of T and an integer t, accepts if there is a sequence of at most t which T can take starting at
c1 and ending at c2.

CANYIELD(c1, c2, t):
If t = 0:
Accept if c1 = c2.
Else if t = 1:
Accept if T can take a single step from c1 to c2.
Else:
For all possible configurations cm of T:
Accept if CANYIELD(c1, cm, bt/2c) and
CANYIELD(cm, c2, dt/2e) both accept.
Reject.
Without loss of generality, we can assume that before T accepts, it writes 0’s on each cell of the tape it used and moves the tapehead to the far left of

the tape. Call this configuration cacc. Let c be the the number of states of T plus the number of symbols in the tape alphabet of T. Then there are at most c f(n) valid configurations of T (because each configuration is at most f(n) long and each thing in the configuration is either a member of the tape alphabet or a state of T). This means T takes at most c f(n) steps on any input on which it terminates, because otherwise it must repeat a configuration and run forever. Let cstart be the start configuration of T on a given input w.

Then T accepts w if and only if:

CANYIELD(c start, c acc , c f(n))

accepts. In addition, we can see that this will take at most O(f(n)) space. This is because each call to CANYIELD requires O(f(n) 2) space to store the encoded cm it chooses plus space for its recursive calls (it will make multiple recursive calls, but only one at any time, so the space for those can be reused). Because we halve t each time, there will be log(c f(n)) = O(f(n)) recursive calls, so the total space need is O(f(n))O(f(n)) = O(f(n) 2).

If f(n) is computable with O(f(n) 2) space, then we can compute c f(n) before our first call to CANYIELD. Otherwise, we can add a size parameter s to CANYIELD, and instead of iterating over all configurations of T, only iterate over configurations which use at most s space. Define cacc,s to be the configuration of T that is the accept state followed by s zeros. Then we can model T using the following algorithm:

For i = 1, 2, 3 . . .

For each configuration c G with length less than i + 1:

If CANYIELD(c1, c G , c i+1, i + 1) accepts:

Accept if c G = c acc,i+1

Increment i and loop.

Reject.

This algorithm accepts if at any time CANYIELD tells us we can reach an accept state. In addition, it rejects if we reach an i such that T cannot yield any configuration of length i + 1. As T never uses more than f(n) space, we will thus reject once i = f(n) + 1, which can only happen when T never reaches an accept state of length f(n), so we know T rejects this input.

We have proven that whenever $f(n) \geq n$, every non-deterministic Turing machine T which requires at most f(n) space on input of length n can be simulated by a deterministic Turing machine which requires O(f(n)) space, so we have shown $NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$.

An important corollary to this is that PSPACE (the set of all languages which can be decided using a Turing machine with polynomial space) is equal to NPSPACE (the set of all languages which can be decided using a non-deterministic Turing machine with polynomial space).

Implications of Savitch's Theorem.

• PSPACE = NPSPACE.
•Nondeterminism is less powerful with respect to space.
• Nondeterminism may be very powerful with respect to time
as it is not known if P = NP.